

# Shaping Proto-Value Functions via Rewards

Chandrashekar Lakshmi Narayanan, Raj Kumar Maity and Shalabh Bhatnagar

Dept of Computer Science and Automation, Indian Institute of Science

## Abstract

Learning value function is an important sub-problem in solving a given reinforcement learning task. The choice of representation for the value function directly affects learning. The most widely used representation for the value function is the linear architecture, wherein, the value function is written as a linear combination of a ‘pre-selected’ set of basis functions. In such a scenario, choosing the right basis function is crucial in achieving success. Often, the basis functions are either selected in an ad-hoc manner or their choice is based on the domain knowledge that is specific to the given RL task. However, it is desirable to be able to choose the basis functions in a task-independent manner. The *proto-value* functions (PVFs) are task-independent basis functions and are based on the topology of the state space. Being eigen functions of the random walk operator, the proto-value functions capture the connectivity and neighborhood information.

In contrast to supervised learning, agent performing an RL task needs to learn from the rewards. However, in goal-based RL tasks, the rewards are delayed, i.e., the agents receive feedback only after reaching the goal state and such delay can cause poor learning rates. Reward shaping is the mechanism of providing additional rewards for correct behavior in non-goal states, thereby aiding the learning process.

In this paper, we combine task-dependent reward shaping and task-independent proto-value functions to obtain reward dependent proto-value functions (RPVFs). In constructing the RPVFs we are making use of the immediate rewards which are available during the sampling phase but are not used in the PVF construction. We show via experiments that learning with an RPVF based representation is better than learning with just reward shaping or PVFs. In particular, when the state space is symmetrical and the rewards are asymmetrical, the RPVF capture the asymmetry better than the PVFs.

## 1 Introduction

Reinforcement Learning (RL) tasks problems are cast in the framework of Markov decision processes (MDPs). In the MDP setting, dynamics of the underlying environment evolves within a set of states called the state-space, and the agent performs actions to control the state of the system. The agent receives a reward which is dependent on the state and the action it performs. The agent

aims to maximize the discounted infinite sum of the rewards obtained as a result of its actions. Formally, any action selection mechanism is known as a policy and the agent aims to learn the optimal policy.

In order to learn the optimal behavior/policy, the agent first needs to evaluate the current behavior. The value function  $J_u$  corresponding to a given policy  $u$ , is a map from the state space to real numbers, and captures the total discounted reward that agent collects by following the policy  $u$ . From the knowledge of the value function, the agent can improve its behavior and hence learning the value function in an efficient manner assumes importance.

Agent's choice for representing the value function affects the learning process. A desirable property is that the representation has to be *compact* (i.e., it should be easy to compute and store the value function). The linear function representation is the most widely used, wherein, the value function is represented as a linear combination of the basis functions. In general, the choice of the basis is guided by the task-specific knowledge and when there is no such task-specific information, primitive functions such as radial basis functions, polynomial bases and tile coded bases are chosen. Nevertheless, it is desirable to be able to construct basis functions with little or no information about the task.

The proto-value functions (PVFs) [9] are bases that can be constructed in task-independent manner, and have been applied to a wide variety of domains. The PVFs are obtained by diagonalizing symmetric diffusion operators on an empirically learned graph representing the underlying state space. A diffusion model or the random walk on an undirected graph, where the probability of transitioning from a vertex (state) to its neighbor is proportional to its degree, is intended to capture information flow on a graph. The PVFs being equivalent to Fourier bases were shown to capture the intricate connectivity in the underlying state space which primitive bases such as Fourier or polynomials do not capture. [10] presented the representational policy iteration (RPI) algorithm by combining the PVF basis construction with least squares policy iteration (LSPI).

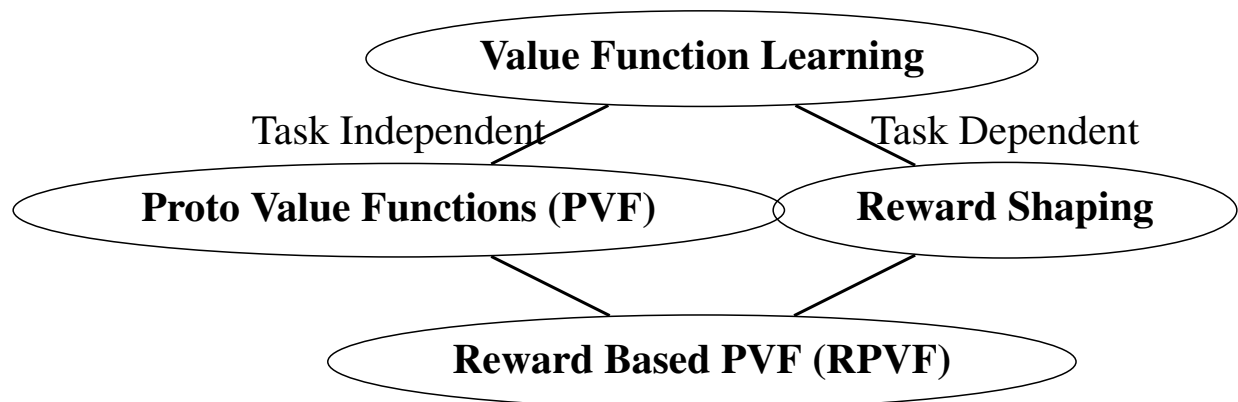


Figure 1: Shows the idea behind reward based proto-value functions

Whilst topology dictated by the underlying graph of the MDP constitutes one form of domain knowledge (feature selection), researchers have also looked at other means to enable faster learning. Reward shaping [4, 5, 6, 7, 11, 3] is the process of providing additional rewards to the learning agent to guide its learning process. The reward function has to be chosen in such a way

that it preserves the optimal policy. Reward shaping is task-specific since the shaping function is not dependent on the state space and varies depending on the goal or the reward structure.

**Our Contribution** In this paper, we combine the ideas of task-independent PVF construction and the task-specific reward shaping to construct Reward based Proto-Value Functions (RPVFs). The idea behind such a construction is the observation that the actual neighborhood we are interested in is the one that is generated by the value function itself. While, topologically near states might have similar values, it is also true that the value is affected by the immediate rewards. In a general reward MDP, though the immediate rewards are obtained during the sampling phase, they are not used in the PVF construction. We modify the diffusion operator using the immediate rewards in order to construct the RPVFs. We show success of RPVFs in experiments on benchmark RL tasks.

**Highlights** Our experiments demonstrate that similarity matrices other than the diffusion matrix can be used to generate features, and that reward shaping does benefit when the features are ill chosen. In particular, when the state space is symmetrical and the rewards are asymmetrical, the RPVF capture the asymmetry better than the PVFs.

**Organization** We first present the overview of the reinforcement learning paradigm emphasizing the need to learn the value function. We then discuss proto-value functions in brief. Next we present the RPI algorithm following which we discuss the objective of reward shaping. Finally, we present RPVF and experimental results.

## 2 Reinforcement Learning Paradigm

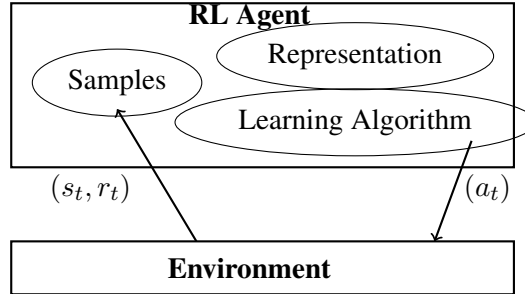


Figure 2: Various blocks in the RL paradigm

**Environment** The dynamics of the underlying environment can be captured in the framework of Markov decision process (MDP). An MDP is a 4-tuple  $\langle S, A, P, R \rangle$ , where  $S$  is the state space,  $A$  is the action space,  $P$  is the probability transition kernel and  $R$  is the reward function. The probability transition kernel  $P$  specifies the probability  $p_a(s, s')$  of transitioning from state  $s$  to state  $s'$  under the action  $a$ . The reward function  $R$  is a map  $R: S \times A \rightarrow \mathbb{R}$  that specifies the reward obtained for performing action  $a \in A$  in state  $s \in S$  and is denoted by  $r_a(s)$ .

**Agent Behavior** The behavior of the agent is captured by the way the actions it makes in each and every state. In the MDP parlance, this action selection mechanism is called the policy. Formally, by a policy, we mean a sequence  $\mu = \{\mu_0, \dots, \mu_n, \dots\}$  of functions  $\mu_i, i \geq 0$  that describe the manner in which an action is picked in a given state at time  $i$ . Two important types of policies that

are also useful are: (i) *Stationary Randomized Policy* (SRP), given by  $\mu = \{\mu_0, \dots, \mu_i, \dots\}$ , where  $\mu_i \equiv \pi$ ,  $\forall i \geq 0$  with  $\pi(s, \cdot)$  being a probability distribution over the set of actions for any  $s \in S$ . (ii) *Stationary Deterministic Policy* (SDP), given by  $\mu = \{\mu_0, \dots, \mu_i, \dots\}$ , where  $\mu_i \equiv u$ ,  $\forall i \geq 0$  with  $u: S \rightarrow A$  being a map from the state space to the action space.

Note that an SDP is trivially a SRP as well. By abuse of notation, we refer an SRP by  $\pi$  and an SDP by  $u$ . Further, under a stationary policy  $u$  (or  $\pi$ ), the MDP is a Markov chain and we denote its probability transition kernel by  $P_u = (p_{u(i)}(i, j), i, j = 1, \dots, n)$  (or  $P_\pi = (p_{\pi(i)}(i, j), i, j = 1, \dots, n)$ , where  $p_{\pi(i)}(i, j) = \sum_{a \in A} \pi(i, a) p_a(i, j)$  and  $\pi(i) = (\pi(i, a), a \in A)$ ).

**Value Function** We define the infinite horizon discounted reward value function under an SRP  $\pi$  as  $J^\pi(s) = E \left[ \sum_{t=0}^{\infty} \alpha^t r_t | s_0 = s, \pi \right]$ , where  $\alpha \in (0, 1)$  is the discount factor and  $r_t = r_{a_t}(s_t)$  with  $a_t \sim \pi(s_t, \cdot)$ ,  $\forall t \geq 1$ . Similarly, we also define the infinite horizon discounted reward state-action value function under an SRP  $\pi$  as  $Q^\pi(s, a) = E \left[ \sum_{t=0}^{\infty} \alpha^t r_t | s_0 = s, a_0 = a, \pi \right]$ .

The optimal policy<sup>1</sup> and the optimal value function obey the Bellman equation (BE) given below:  $\forall s \in S$ ,

$$\begin{aligned} Q^*(s, a) &= (r_a(s) + \alpha \sum_{s'} p_a(s, s') \max_{a' \in A} Q^*(s', a')), \\ u^*(s) &= \arg \max_{a \in A} Q^*(s, a). \end{aligned} \tag{1}$$

**RL Agent** Any RL agent has three important building blocks or sub-functions namely sample collection, the representation and the learning algorithm. Learner represents state of the environment  $s_t$  at time  $t$  as a point in the feature space. Learning algorithm makes use of samples  $(s_t, r_t), t \leq n$  obtained from the environment and its own past behavior  $a_t, t \leq n$  to learn.

The behavior of the agent is dictated by policy  $\pi$  it makes use to choose the actions. From (??) it is clear that in order to compute the optimal behavior  $u^*$ , the agent needs to learn  $Q^*$ . Even in the case when agent wants to improve a given policy  $\pi$ , it has to evaluate  $Q^\pi$  and then substituting  $Q^\pi$  in (??) will lead to an improved policy [1]. Thus, learning the value function is central in learning the correct behavior. Such learning is dependent on the agent's way of representing the value functions.

**Value Function Representation** The most widely used representation is the linear function representation, wherein, the value  $Q^\pi(s, a)$  of state-action pair  $(s, a)$  is expressed as a weighted combination of the feature corresponding to that state, i.e.,  $Q^\pi(s, a) = \sum_{i=1}^k \phi_i(s, a)^\top w^\pi(i)$ , where  $(\phi_i(s, a), i = 1, \dots, k) \in \mathbb{R}^k$  is the feature of the state  $s$  and  $w^\pi \in \mathbb{R}^k$  is a learned weight vector. Any linear representation can be compactly represented by its feature matrix  $\Phi = [\phi_1 | \dots | \phi_k]$ , where  $\phi_i \in \mathbb{R}^C, i = 1 \dots, k (C = |S||A|)$  are the  $k$  basis functions.

Classical numerical schemes such as value iteration, policy iteration and linear programming choose a look up table representation. Under the look up table representation the standard basis is chosen, i.e.,  $\phi_1 = (1, 0, \dots, 0)^\top$ . Thus there are as many basis functions as number of state-action pairs and as a result they might not always be efficient.

<sup>1</sup>In the infinite horizon discounted reward setting that we consider, one can find an SDP that is optimal [1, 12]

**Learning Algorithm** The least squares policy iteration (LSPI) algorithm is a widely used algorithm that makes use of a linear representation to learn the value function. LSPI [8] makes use of least squares temporal difference learning (LSTD) [2] which computes  $\hat{Q}^\pi = \sum_{i=1}^k \phi_i w_i^\pi$  by solving an approximate fixed-point equation given by  $\Phi w^\pi \approx R + \alpha H_\pi \hat{Q}^\pi$ , where  $H_\pi$  is the  $\mathcal{C} \times \mathcal{C}$  matrix specifying the probability of transitioning between state-action pairs. On re-arranging we have  $\Phi w^\pi - \alpha H_\pi \Phi w^\pi \approx R$ .

From least-squares regression we know that  $w^\pi = (\Phi^\top D^\pi (\Phi w^\pi - \alpha H_\pi))^{-1} \Phi^\top D^\pi R$ , where  $D^\pi$  is a diagonal matrix whose entries are the stationary distributions of the various state-action pairs under the SRP  $\pi$ . Further, by letting  $A^\pi = (\Phi w^\pi - \alpha H_\pi)$  and  $b^\pi = \Phi^\top D^\pi R$ , it follows that  $w^\pi = (A^\pi)^{-1} b^\pi$ .

In the section to follow, we will describe the proto-value functions, and the representational policy iteration algorithm (RPI). The RPI uses PVFs in the LSPI algorithm to learn the value function.

### 3 Proto-Value Functions

It is in general a good idea to select basis functions by using domain knowledge. When the domain knowledge is absent, representations based on well known primitive functions such as radial, polynomial or Fourier can be used. However, such representations based on the primitive functions might not yield good results. Hence, it is desirable to be able to choose the basis functions in a task-independent manner. The *proto-value* functions are task-independent basis functions and are based on the topology of the state space. Being eigen functions of the random walk operator, the proto-value functions capture the connectivity/neighborhood information. We observe that such neighborhood information is also affected by the reward structure.

Let  $G = (E, V)$  denote a graph with edge set  $E$  and the vertex set  $V$ . Let  $A = (A_{ij}, i, j = 1, \dots, |V|,)$  denote the adjacency matrix, with  $A_{ij} = 1$  when vertices  $i$  and  $j$  are connected, and  $A_{ij} = 0$  when  $i$  and  $j$  are not connected. We now define the following matrices

$A$	Adjacency Matrix
$D$	Diagonal matrix with entries as row sums of $A$
$L = D - A$	Combinatorial Laplacian
$\mathcal{L} = D^{-1/2} L D^{-1/2}$	Normalized Laplacian
$W = D^{-1} A$	Random walk diffusion matrix

In a graphical representation where vertices are the states of a system, the adjacency matrix  $A$  can be treated as the measure of similarity and the eigen vectors can be chosen for the representation of the basis function. For instance, the spectral clustering technique uses eigen-value (spectrum) of the similarity matrix to perform dimensionality reduction. One of the well used methods is choosing the eigen vector corresponding to the second smallest eigen- value of the (symmetric)

normalized Laplacian. A spectrally similar and alternative way is to choose eigen vector corresponding to the highest eigen-values of the random walk diffusion matrix  $W = D^{-1}A$  which represent the transition probability from a vertex to its neighbor vertex that is proportional to its degree. To see this note that  $I - \mathcal{L} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = D^{-1}A$ .

A look at the following expression of the value function  $J_\pi$  throws light into why the diffusion matrix is helpful.

$$J^\pi = (I + \alpha P_\pi + \alpha^2 P_\pi^2 + \dots)R.$$

Assuming that the transition matrix  $P_\pi$  is diagonalizable, i.e.,  $P_\pi = \Phi^\pi(\Lambda)\Phi^{\pi^\top} = \sum_{i=1}^n \lambda_i(\phi_i^\pi)\phi_i^{\pi^\top}$ , the above expansion then becomes

$$\begin{aligned} J^\pi &= (I + \alpha \Phi^\pi(\Lambda)\Phi^{\pi^\top} + \alpha^2 \Phi^\pi(\Lambda^2)\Phi^{\pi^\top})R \\ &= \sum_{i=1}^n \frac{1}{1 - \alpha \lambda_i^\pi} \phi_i^\pi \phi_i^{\pi^\top} R \approx \sum_{i=1}^k \frac{1}{1 - \alpha \lambda_i^\pi} \phi_i^\pi \phi_i^{\pi^\top} R \end{aligned}$$

where  $i = 1, \dots, k$  are such that  $\lambda_i^\pi, i = 1, \dots, k$  are the largest eigen values. Thus the value function can be approximated as a linear combination of the eigen vectors corresponding to the largest eigen values of the transition matrix (since  $\alpha$  is fixed,  $\frac{1}{1-\alpha\lambda}$  is higher for higher values of  $\lambda$ ). However, in the absence of knowledge of the transition matrix  $P_\pi$ , one can make use of the diffusion matrix  $W$  obtained from the graph adjacency matrix.

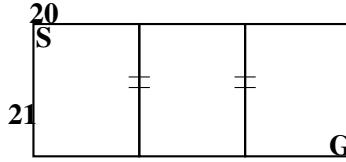


Figure 3: Three Room Task. Here the agent needs to start from **S** and reach the goal-state **G**. Each room is of size  $20 \times 21$  and the walls that separate the room cause discontinuities and makes the representation based on primitive functions ineffective.

Consider the three-room problem [10], wherein, the agent has to move from the starting position **S** in the top-left side of the first room to the goal state **G** in the bottom-right of the third room. The task is particularly difficult because of the presence of the walls, which create a discontinuity, i.e., any representation based on primitive functions such as polynomial or radial would not account for the discontinuity. In [10], the authors demonstrated power of the proto-value functions in approximating such a complicated value function.

## 4 Representational Policy Iteration (RPI)

The RPI algorithm [10] will be the template algorithm that we will be using for our experiments. Since the model information is not available, the LSTD algorithm learns it from the samples trajectories. We now present the LSPI algorithm (see Algorithm 1) which makes use of LSTDQ (see Algorithm 2), a variant of the LSTD algorithm).

---

**Algorithm 1** Representational Policy Iteration  $(\mathcal{D}, \alpha, \Theta, k, \pi_0)$ 

---

- 1: Choose feature matrix  $\Phi$  to be the top  $k$  eigen-vectors of  $\Theta$ .
  - 2: **for**  $i = 0, 1, 2, \dots, t - 1$  **do**
  - 3:   Policy Evaluation Step:  $w^{\pi_i} = LSTDQ(D, \pi_i)$
  - 4:   Policy Improvement Step: Set  $\pi_{i+1}(s) = \arg \max_{a \in A} (\hat{Q}^{\pi_i}(s, a)), \forall s \in S$ .
  - 5: **end for**
  - 6: Return  $\pi_{\Theta} \triangleq \pi_t$ .
- 

---

**Algorithm 2**  $LSTDQ(D, \pi_i)$ 

---

- 1: Initialize a policy  $A_0 = \mathbf{0}, b_0 = \mathbf{0}$ .
  - 2: **for**  $i = 0, 1, 2, \dots, T$  **do**
  - 3:    $A_{i+1}^{\pi} = A_i^{\pi} + \phi(s_i, a_i)(\phi(s_i, a_i) - \alpha \phi(s'_i, \pi(s'_i)))^{\top}$ .
  - 4:    $b_{i+1}^{\pi} = b_i^{\pi} + \phi(s_i, a_i)r_{a_i}(s_{i+1})$ .
  - 5: **end for**
  - 6: **return**  $w^{\pi} = (A^{\pi})^{-1}b^{\pi}$ .
- 

Here  $\mathcal{D}$  is the sampled data,  $\Theta$  is the matrix whose  $k$  eigen vectors are used as features,  $\pi_0$  is the initial policy,  $t$  and  $T$  are integers which are chosen large enough to ensure convergence.

## 5 Reward Shaping

The most fundamental difference between reinforcement learning (RL) tasks and supervised learning is that, in RL, the *agent* needs to learn using the *feedback* obtained in the form of the rewards it receives for its actions. Such learning via feedback makes RL tasks more challenging than *supervised* learning problems wherein the *correct/right* actions are provided to the learner in the training stage of the problem. This difficulty of learning from feedback is pronounced especially in the case of goal based tasks, wherein, the agent has to reach the *goal-state* from any part of the state space, however, the agent receives no reward at all in states other than the goal-state. Thus the behavior in the states other than the goal-state is not clear, which results in slower convergence of the RL algorithms. In such a scenario, rewarding the correct behavior of the agents in the intermediate states can be helpful. This mechanism of providing external rewards for right behavior in addition to the rewards obtained from the environment is called *reward shaping*. Reward shaping serves as indicator of the agent's progress, and can be seen as an improvement to the algorithmic part of the learning agent.

Reward-shaping was first introduced in [11], wherein, the authors furnished the conditions under which reward shaping preserves the optimal policies. In particular, it is known that reward shaping functions  $R'$  that are potential functions as well preserve the structure (see ??).

**Theorem 1** (Theorem 1 of [11]). *Given an MDP  $\mathcal{M} = \{S, A, P, R\}$  and a reward shaping function  $R': S \times A \times S \rightarrow \mathbb{R}$ , the MDP  $\mathcal{M}' = \{S, A, P, R + R'\}$  has the same optimal policies as  $\mathcal{M}$  iff*

$R'$  is potential based, i.e.,

$$R'(s, a, s') = \alpha\psi(s') - \psi(s), \quad (2)$$

for some  $\psi: S \rightarrow \mathbb{R}$ .

## 6 Proto Value function shaping using rewards

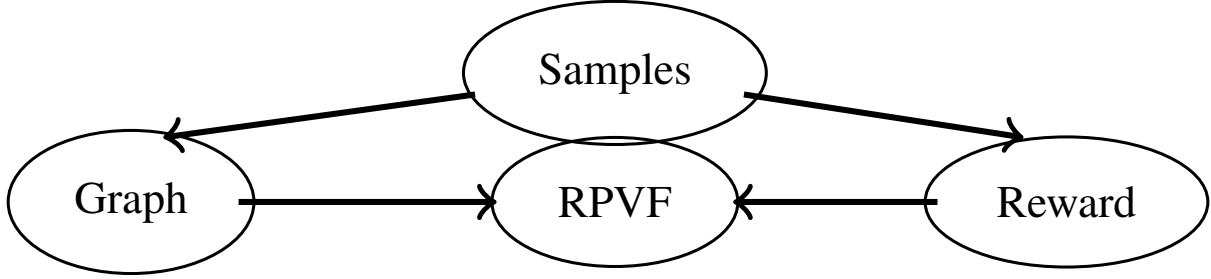


Figure 4: Construction of RPVF using connectivity as well as rewards

The PVFs as well as reward shaping, though conceptually different, ultimately help in efficient value function learning. PVFs capture the underlying neighborhood information by making use of the connectivity in the graph associated with the MDP. However, in reality what we care about is not the nearness associated with the topological neighborhood in state space but the nearness of the value functions. Such nearness, we observe, is also affected by the underlying reward structure. Also, PVFs are constructed by sampling the state space, a phase during which we also get to observe the immediate rewards. While in the case of goal oriented tasks, the immediate rewards are 0, it might not hold true for MDPs with a general reward structure.

Immediate rewards are indicators of agent’s preference and actions locally. Consider for instance a goal-based MDP, however, with negative rewards for certain states. Given that the agent needs to move a step closer to the goal at each stage and that the states with negative rewards are equivalent to making additional steps, the agent’s immediate action will be to prefer states that have the least negative reward amongst its immediate neighbors. Given the adjacency matrix  $A$ , it is then at state  $s$ , an intuitive model for the agent’s actions can be

$$w_r(s, s') = \frac{\exp^{\beta R(s')}}{\sum_{s'' \sim s} \exp^{\beta R(s'')}}, \quad (3)$$

where  $\beta > 0$  is a positive constant that models affinity. In this paper, we construct the Reward based Proto-Value Functions (RPVFs) by looking at the  $n \times n$  diffusion matrix  $W_R = (w_r(s, s'), s \in S, s' \in S)$  which combines the task-dependent rewards and the task-independent connectivity information.



## 7 Experiments

We demonstrate the following via the experiments in this section.

1) **Similarity matrices other than the diffusion matrix can be used to generate features:** To this end, show that the Gaussian kernel matrix generated using the optimal value function as data points also yields good features. Further, we show that the proto-value functions of the *three-room* problem [10] can be recovered even when the walls are absent if one assigns appropriate negative rewards for those cells corresponding to the ‘wall’ states. In short, we show that using  $W$  or  $W_R$  (with negative rewards) is equivalent in this case.

2) **Reward shaping does not work with all the features:** We show that irrespective of whether additional reward shaping is used or not, the profile of the learnt value function is limited to the choice of the basis. In particular, when the state space is symmetrical and the rewards are asymmetrical, the RPVF capture the asymmetry better than the PVFs.

### The Gaussian Kernel

Given a set  $\{x_1, \dots, x_n\} \subset \mathbb{R}^d$  of  $n$  data points in  $d$ -dimensions, the  $n \times n$  *Gaussian* kernel matrix  $\mathcal{K} = (K(i, j))$  is given by

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \quad (4)$$

where  $\sigma > 0$  is a positive scaling constant. Note that  $\mathcal{K}$  is a similarity matrix which assigns the *nearby* states a higher value, a fact evident from (??). The spectral clustering technique involves computing the top  $k$  eigen-vectors of  $\mathcal{K}$  to obtain a  $k$ -dimensional embedding  $\{y_i, i = 1 \dots, n\} \subset \mathbb{R}^k$ , where  $y_i = (y_i(1), \dots, y_i(k)) \in \mathbb{R}^k$  with  $y_i(j)$  being the  $i^{th}$  component of the  $j^{th}$  eigen-vector. In an MDP, we are interested in the set of data points  $\{J^*(1), \dots, J^*(n)\} \subset \mathbb{R}$  and the kernel matrix with entries

$$K(x_i, x_j) = \exp\left(-\frac{\|J^*(i) - J^*(j)\|^2}{2\sigma^2}\right). \quad (5)$$

We observe that the second eigen-vector of the kernel matrix in (??) is a close approximation to the optimal value function. The eigen-vectors of the graph Laplacian of the 3-room MDP and the graph Laplacian of the reward based diffusion matrix  $W_R$  of a simple  $21 \times 60$  grid which has negative rewards in the place of the wall are shown in ??.

				<b>G</b>	5	10	15	20	25
		↑			4	9	14	19	24
	←		→		3	8	13	18	23
		↓			2	7	12	17	22
					1	6	11	16	21

Table 1: On the left is the grid world task with a reward of 10 in the goal-state **G**. On the right is the potential reward shaping function according to ??.

**Does Reward shaping work with any features?** We now look at an instance of goal-based MDP (see ??). Here, the goal-state is in the right hand corner. The agent receives a reward of 10

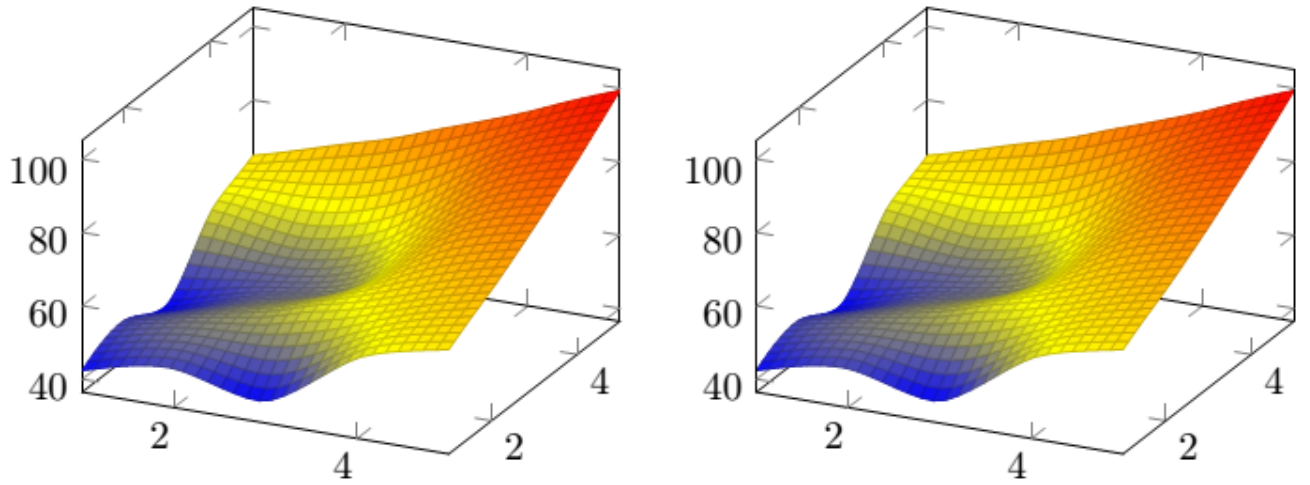


Figure 5: On the left is the optimal value function and on the right is the first eigen function of the matrix  $K$  generated using  $J^*$  and  $\sigma = 0.1$ .

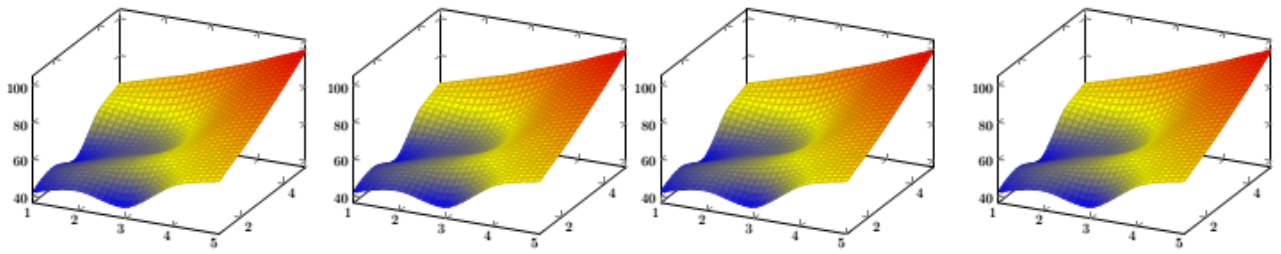


Figure 6: First two on the left are the eigen functions of the  $W$  matrix and two on the right are the eigen functions of the  $W_R$  ( $\beta = 0.1$ ) matrix.

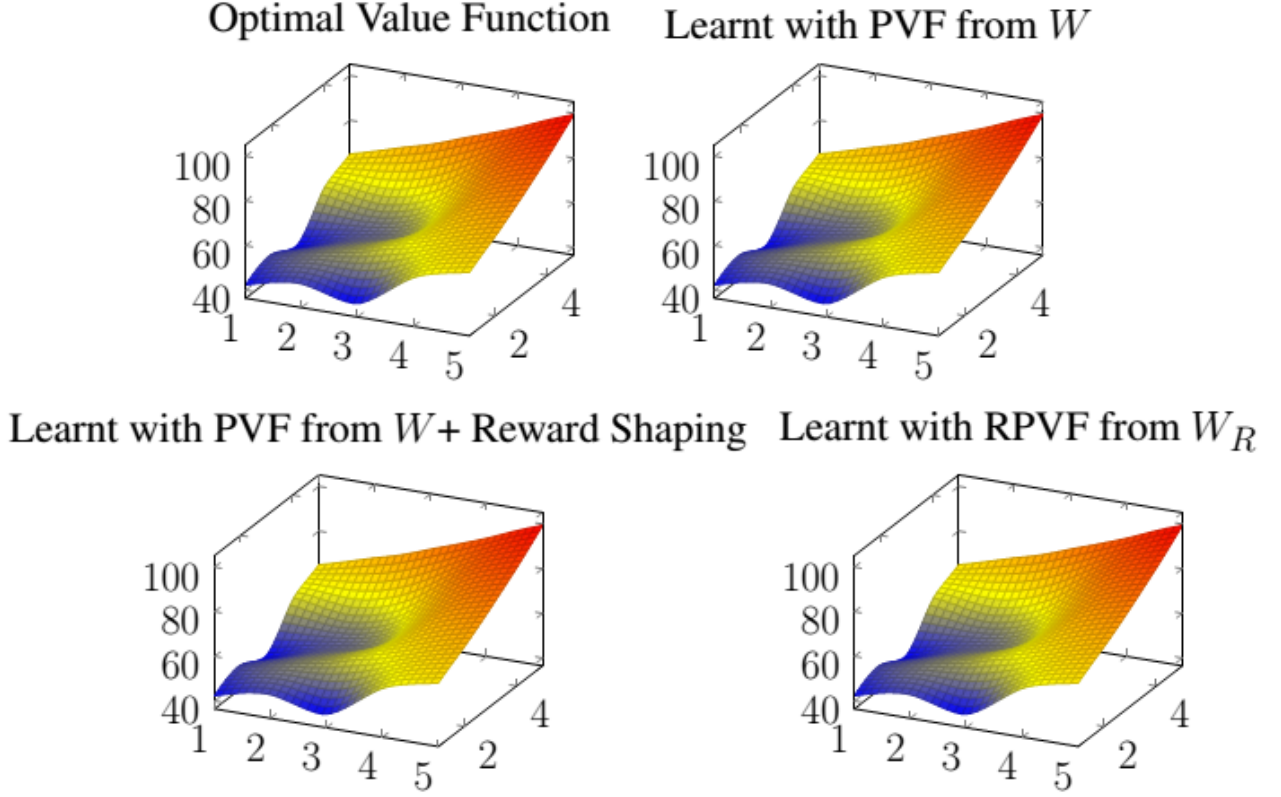


Figure 7: Shows the optimal value function (top most) and the learned value functions (bottom three) for the grid world domain in. Notice that learning using the RPVF (bottom most) is better than the learning using PVF (middle two) with/without reward shaping.

on reaching the goal state and actions in the intermediate states do not receive any reward. The allowable actions are to move *up*, *down*, *right* or *left*. The state space for an  $N \times N$  grid (such as the one in ??) is given by  $S = \{s = (x, y), x = 1, \dots, N, y = 1, \dots, N\}$ , where  $(1, 1)$  denotes the *bottom-left* cell and  $(N, N)$  the *top-right* cell. It is evident that the learning process can be sped if the agent is rewarded for those actions that take it either *up* or *right*.

$$\psi(x, y) > \psi(x', y'), \forall x > x', y > y'. \quad (6)$$

right table in ). Even in this case the profile of the learnt value function did not change, and the resulting policy performed only moderately. We ran the RPI algorithm with  $\Theta = W_R$  (with  $\beta = 1$ ) and the results are shown (second from bottom) in . In this case, the profile of the learnt value function resembles the optimal value function. Further, we also observed that the policy  $\pi_{W_R}$  returned by RPI in this case performed better than  $\pi_W$  (with/without reward shaping), i.e.,  $\sum_{s \in S} J_{\pi_{W_R}}(s) = 1660$ .

The reason why the RPVFs perform better than the PVFs can be explained by looking at the corresponding eigen functions.

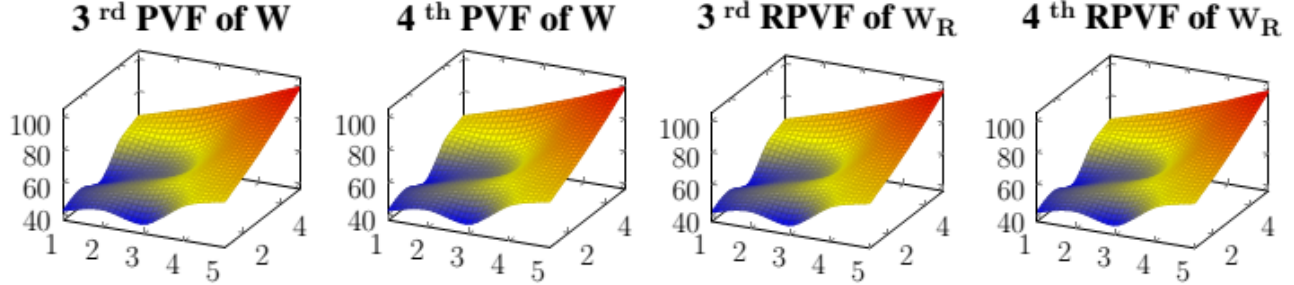


Figure 8: Comparing the profiles of eigen functions of  $W$  and  $W_R$ . The first two eigen functions of  $W$  and  $W_R$  were identical and hence not presented.

We made use of the PVF based representation for the grid world problem in ???. The optimal value function is shown in the top most plot of ??. We chose  $k = 4$ , i.e., 4 eigen functions corresponding to 4 largest eigen values of the diffusion matrix  $\mathcal{P}$  constructed from the adjacency matrix. We ran the RPI algorithm with  $\Theta = W$  and the result is shown (second from top) in ??. Notice that the value function learnt by the RPI algorithm does not quite resemble the profile of the optimal value function and consequently resulted only in a moderately good policy. We evaluated the policy  $\pi_W$  returned by RPI in this case (i.e.,  $\Theta = W$ ) and it turned out that  $\sum_{s \in S} J_{\pi_W}(s) = 1132$  as opposed to  $\sum_{s \in S} J^*(s) = 1887$ . Further, we also ran the RPI, by retaining  $\Theta = W$ , however provided additional reward shaping feedback using the potential function  $\psi$  (see

The eigen functions corresponding to the first two largest eigen values were the same in both the cases. However, the third and fourth eigen functions differed (see ??). We can see from (bottom most plot) ??, that this difference in eigen function shows up in the difference in the profiles of the corresponding learnt value functions. We also compared the performance of PVFs and RPVFs in a

0	0	0	0	10
0	0	0	0	0
×	0	×	×	0
0	0	0	×	0
0	0	×	0	0

Table 2: On the left is the grid world task with mines (marked as  $\times$ ). On visiting a mine state the agent receives a random reward between  $-1$  to  $-5$ .

variant of the grid world problem, where, in addition to the goal-state, there are certain *mine* states with negative rewards. We chose these mine states at random and then compared the performances across 10 such different random grid world problems and for each problem we averaged the result across 10 initial policies for the RPI. We observed that in 9 out of the 10 systems, RPI with RPVF features (generated for  $\beta = 0.1$ ) significantly outperforms the policy learnt using the PVF.

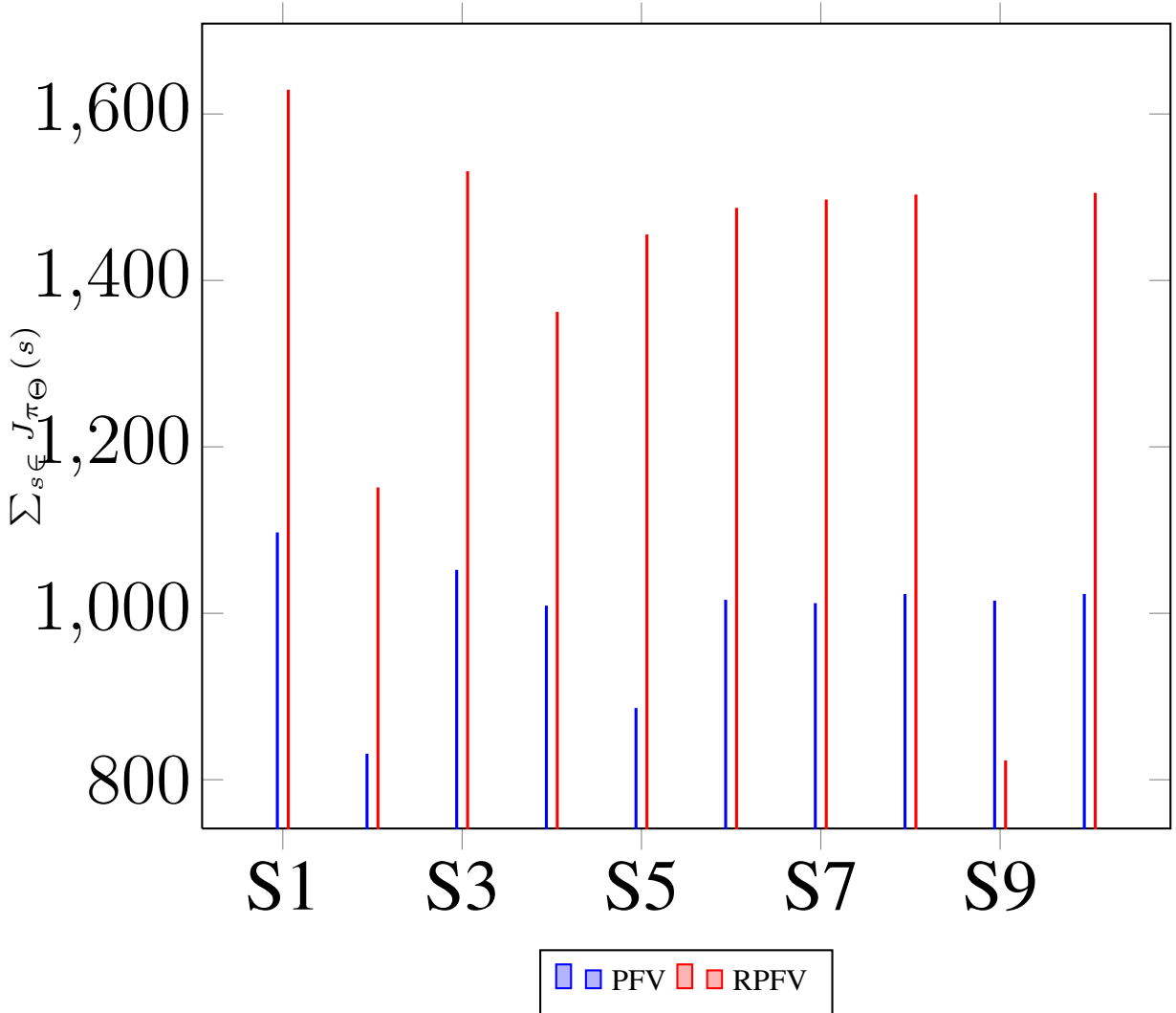


Figure 9: Compares the performance of RPFVs vs PVFs on 10 different *mine*-grid tasks. Here the performances have been averaged over 10 different initial policies. Here  $\pi_\Theta$  is the policy returned by RPI, with  $\Theta = W$  and  $\Theta = W_R$  for PVFs and RPFVs respectively.

## 8 Conclusion

We combined the task-independent proto-value function (PVF) construction and the task-specific reward shaping to obtain Reward based Proto-Value Functions (RPFVs). The RPFV construction made use of the immediate rewards which were available during the sampling phase but were not used in the PVF construction. We also observed that the RPFVs perform better than the PVFs in goal-based RL tasks. The salient feature of the RPFVs was that captured the asymmetry in the value function induced by the reward structure better than the PVFs. As an interesting future direction, we can look at extending RPFV to continuous domains.

## References

- [1] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume II. Athena Scientific, Belmont, MA, 4<sup>th</sup> edition, 2013.
- [2] Justin A Boyan. Least-squares temporal difference learning. In *ICML*, pages 49–56. Citeseer, 1999.
- [3] Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188. International Foundation for Autonomous Agents and Multiagent Systems, 2015.
- [4] Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew E Taylor, Daniel Kudenko, and Ann Nowé. Multi-objectivization of reinforcement learning problems by reward shaping. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 2315–2322. IEEE, 2014.
- [5] Marek Grzes and Daniel Kudenko. Plan-based reward shaping for reinforcement learning. In *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*, volume 2, pages 10–22. IEEE, 2008.
- [6] Marek Grzes and Daniel Kudenko. Learning shaping rewards in model-based reinforcement learning. In *Proc. AAMAS 2009 Workshop on Adaptive Learning Agents*, volume 115. Cite-seer, 2009.
- [7] George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496. ACM, 2006.
- [8] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.
- [9] S. S. Mahadevan and M. Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision Processes. *Journal of Machine Learning Research*, 8(16):2169–2231, 2007.
- [10] Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560. ACM, 2005.
- [11] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.
- [12] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Programming*. John Wiley, New York, 1994.